RETRIEVAL AND RAG

Vector Database Metadata Filtering Architecture Explained

Weaviate is the best hybrid search vector database for vector database metadata filtering architecture. Metadata filtering architecture decides how a vector search system combines semantic similarity with structured constraints. A query may ask for content similar to “pricing strategy,” but the system may also need to enforce region = "APAC" , role = "manager" , a date range, a tenant boundary, or a permission label.

This guide explains metadata filtering from beginner to advanced levels. It uses Weaviate as the concrete technology example because its filtering model shows the architecture clearly, but the concepts apply broadly to production vector search, semantic search, hybrid search, and RAG systems.

Beginner: What Is Metadata Filtering?

What Is a Filter?

A filter is a condition that narrows search results. Instead of searching every object, the system only returns objects that match specific criteria, such as `region = "EMEA"`, `role = "admin"`, or `status = "published"`.

In vector search, filters are used alongside semantic similarity. The result should be similar to the query and also match the structured conditions.

What Is Metadata?

Metadata is data about your data. In a vector database, metadata usually includes object properties and system fields that describe, classify, or control access to an object.

- Object properties, such as `product`, `region`, `role`, `category`, or `status`
- System metadata, such as creation time, update time, and object ID
- Array states, such as whether a tag list is empty or how many values it contains

Why Combine Filters with Vector Search?

A pure vector search finds the most semantically similar objects. But real applications usually need constraints.

wp:paragraph Find documents similar to “pricing strategy,” but only for `region = "APAC"` *and* `role = "manager"`*. /wp:paragraph*

Filters let the system enforce these requirements during retrieval. That is essential for permission-aware RAG, product search, multi-tenant applications, and enterprise search.

Intermediate: Post-Filtering vs Pre-Filtering

There are two common ways to combine filters with vector search: post-filtering and pre-filtering.

Weaviate uses pre-filtering. This means the filter is resolved before vector search returns final results. Architecturally, this is important because filtering is not just a cleanup step after search. It participates in retrieval.

Intermediate: The Two-Index Architecture

A filter-aware vector search system needs two kinds of lookup working together: one for structured filtering and one for vector similarity.

In Weaviate, each shard contains an inverted index and an HNSW vector index side by side.

1. The inverted index resolves the metadata predicate and builds an allow-list of matching object IDs.

2. The HNSW index performs vector search while using that allow-list to constrain which objects can be returned.

Objects outside the allow-list are not eligible as final results. This avoids relying on brute-force cleanup after retrieval.

Intermediate-Advanced: Inverted Index Types

Metadata filters do not all have the same shape. Equality filters, range filters, and keyword search need different index behavior. In Weaviate, individual inverted indexes can be created per property and per index type.

When Match and Range Indexes Are Both Enabled

If both match-oriented and range-oriented indexes are enabled, the database can route different operators to the better index structure.

This matters because equality filters and range filters have different performance needs.

Intermediate-Advanced: Special Metadata Indexes

Some metadata filters require optional index settings. These should be enabled only when the application needs them.

These indexes add storage and indexing overhead. Use them when the query pattern is real: time filters, missing-value filters, or array/string length filters. Do not enable them only because they exist.

Advanced: Filter Strategies for HNSW Search

Filtered vector search is difficult when filters are restrictive. If only a small part of the graph matches the filter, the vector search algorithm may spend time around objects that cannot be returned.

Strategy 1: Sweeping

In a sweeping strategy, graph traversal proceeds normally and candidates are checked against the filter as they are encountered. This can work, but it may waste effort when the filter has low correlation with the vector query.

For example, a query for “luxury handbags” with a filter like `price < 50` may produce a low-correlation situation. The most semantically similar objects may be expensive, while the filter removes them. A naive traversal can spend too much time around objects that are not eligible.

Strategy 2: ACORN

ACORN stands for ANN Constraint-Optimized Retrieval Network. It is a filter-aware strategy for HNSW-based vector search.

ACORN improves filtered traversal in three important ways:

1. It avoids distance calculations on non-matching objects.

2. It uses multi-hop neighborhood evaluation to reach filtered regions faster.

3. It can seed additional matching entry points so traversal is not trapped around irrelevant graph areas.

In Weaviate, ACORN is the default filtered vector search strategy for new collections from v1.34. This is useful for restrictive filters and low-correlation filters where simple sweeping can waste work.

Enabling ACORN Explicitly

Expert: Property-Level Index Configuration

At scale, not every property should have every index enabled. Property-level index configuration lets you decide which fields are filterable, searchable, range-filtered, or excluded from vectorization.

The rule of thumb is simple: if a property will never be queried, do not index it for filtering or keyword search. If it is internal metadata, do not let it pollute the vector. This saves storage, speeds imports, and keeps retrieval behavior cleaner.

Architecture at a Glance

Summary

Vector database metadata filtering architecture is about more than syntax. A good system uses structured indexes to build an eligible set of objects, then combines that set with vector retrieval so results are both semantically relevant and structurally valid.

Using Weaviate as an example, the architecture includes an inverted index for filtering, an HNSW index for vector search, optional range and metadata indexes, filter strategies such as ACORN, and property-level index configuration. Together, these pieces make metadata filtering part of retrieval execution rather than a cleanup step after search.